CH8 Level Change

8 What Do We Mean By Level Change

8.1 Working on Different Levels

The H-shaped building I worked in was built on the side of a hill, with the front half of the H half a floor higher than the rear. From my desk in the rear, Human Factors was on the level above me and Maths Modelling half a level up at the front. I never worked with Chip manufacture - below me on the ground floor - nor Materials, who were on the same level as Maths Modelling. Friction Stir Welding, the chicken gun and the self-driving car were in different buildings entirely.

In a building, the term "level" references vertical stacking. In the world of ideas, "going up a level" typically means moving to a more abstract discussion, However, "going up a level" can be used as a cover for magical thinking - statements of the form 'consciousness emerges at a higher level than the mechanisms of the brain' or 'free will appears at a higher level than computer code'. I cannot believe that simply changing the way we talk about something can have any influence over chains of causality, although I will not pursue that argument here. Rather, I want to model what a change of level actually means, and do so in terms of connecting different "levels" of a knitwork. This will detect when a change of level also draws in additional knowledge or abstracts away some of the knowledge used. This has a practical application when building complex systems, where one must match each developer to the problem for which they have the required knowledge.

For example, one of my last R&D proposals was for "Smart Cities" [1], turning the vague general idea of "connecting all the embedded sensors" to an integrated whole in order to help manage city-wide traffic flows, power consumption, and so on. My particular spin was to re-interpret an existing multi-level model in order to break one very large problem into many smaller, connected problems. The problems included identifying standard interfaces for sensors, signal processing to smooth out spikes in the signals, modelling whole systems (for example, to relate traffic flows to traffic light settings), and so on, up to machine learning to predict the evolution of the city (for example, how drivers find new rat runs to get round traffic calming measures). Without such a model, the many providers of the components needed would develop incompatible systems, reinventing the "islands of automation" scenarios of the 1990s, or the incompatible service stacks found in the"everything is a service" fashion of the 2010's.

At its core, "going up a level" is a human factors decision - how can we organise a complex problem into a set of smaller, simpler problems so we can get our head around them one at a time. I can explain how to calculate 1+1 = 2 in terms of logic gates - only a couple of levels to connect together. But I doubt I could describe how to merge real-time traffic data into route planning in terms of quantum effects in semiconductor junctions - too many levels, too much detail, too much knowledge for a single person - I'd need some help.

As a geek, I will focus on formalising "going up a level", starting by reducing the vagueness in my description of a knitwork. In step one I define a morphism - a shape-preserving mapping - that extracts a topic from the whole, vague cloud of knowledge into something small and precise, but at the cost of creating "not-quite-a-knitwork". Step two is to define a morphism from one NQA-knitwork to another, higher level one. Then step three is to try this out on some small examples. What will emerge (spoiler alert) will be several different kinds of level change, some of which will view the lower level through a lens of additional knowledge - knowledge that was not part of the original problem. Then, in the next chapter, we will try this out on Smart Cities, an industrial scale

problem.

However, as my old H-shaped research centre has been closed down, you will only get my geek perspective - I can no longer go up a level to talk to Human Factors, but instead go down the hill to feed my ducks.

8.2 Vague to Precise (but Not-Quite-A-Knitwork)

The definition of a knitwork - facts used intelligently, and inferences driven by knowledge - has so far been left rather nebulous. But to compare two "levels" in a knitwork, we need to map out the areas of knowledge and intelligence we will be comparing. And "map" here is a useful analogy, both recalling the mathematical concept of a mapping, and pointing to the variety of ways we map a landscape. The latter range from ancient theological maps which put Jerusalem at the centre of the world, through the geographic approximations of the London A-to-Z, to the abstracted connections of the London tube map [2]. The way we construct a map depends on what we want to learn from it.

And what we want to learn is how to describe the integration of knowledge and intelligence in particular situations. Start with an analogy: the Japanese dish ramen [3] - slices of meat, half eggs, shrimps and vegetables for facts, with noodles streaming between them for inference, all in a broth providing a tasty context. Now imagine the cloud of knowledge as a large swimming pool filled with ramen. One could not drink it all down at once, but must dip in a bowl in to isolate a digestible topic. When scooping out a bowl full, many noodles will slide over the edge of the bowl and back into the pool - many of the connections of our bowl of ramen to the rest of the swimming pool will be lost. But it was these wider connections that are characteristic of a knitwork, so, being isolated from the rest of the swimming pool, our bowl contains not-quite-a-knitwork.

But this is where our intention - to describe a complex integration environment - helps us to cheat. An information model will link the data involved - the sensor outputs, outputs of computational algorithms, the databases - to the knowledge used. We can start by using an information model as a proxy - as a map of - the knitworks involved. Shrimps being mapped to entities, noodles to inferences, broth to context.

For simplicity, separate out the discussion into three layers: at the bottom, an information model describing the things floating in the soup; at the top a library of inferences - noodles going from one place to another; while in the middle a patch board (like the old manual telephone switch boards) [4] connecting the input knowledge via the inferences to the output knowledge - OK, the ramen analogy is not up to the job, but we can still use the "bowl of ramen" as a way of saying that the scope of what is represented is bounded, and has lost its connections to the broader knowledge ecosystem, so it is not-quite-a-knitwork [5].

Note that the model has three layers, not three levels. The layers are a means of separating out - of organising - different aspects of the map, as if we took a map of the countryside and created semi-transparent overlays for field usage (pasture, pigs, potatoes), ownership and earnings per field. One could put all the overlays on the same map - squash the three layers into a single one - but we will suffer information overload, and it is easier to track what is going on by separating out the layers. How layers differ from levels will be made clear later in the chapter.

Consequently, our map of an isolated knitwork - a bowl of ramen's worth of knowledge - looks rather like an information model with overlays for inferences and their uses. To make the model

more coherent, the only inferences included are those that start and end with the facts described by the information model. However, this doesn't have the depth that a proper knitwork needs - it doesn't have the knowledge to say how and why an inference applies to a particular set of facts, nor the intelligence to apply a different sort of inference if the context changes. But it will have to do for now.

8.3 Morphisms between Knitworks

Roughly, a morphism - from the Greek "having a form" - is a mapping **from** one algebraic structure **to** another which preserves "the shape" of the structure [6]. For an example, take two algebraic structures, the integers and the modulo 12 group (clock maths): we can map the number 4 to number 4, but also map 16 (modulo 12) to 4. The integer operation 4+4 = 8 maps to the same sum in the modulo 12 group, while 8 + 8 maps the result 16 to its equivalent 4 (modulo 12). That is, an algebraic structure is what you get when you combine a set of tokens with a set of operations - the data types of chapter 2 - and a morphism is a mapping that allows one to use the result in the "from" structure to predict the result in the "to" structure. At which point use chocolate biscuits to tempt the mathematicians back into their lairs before they start explaining the differences between an isomorphism and a homomorphism.

As befits morphisms involving knowledge soup, the morphisms we will use will be left a little vague. Start with the morphism that extracts a small knitwork from the vast cloud of knowledge - dips the bowl into the swimming pool. First, it delimits the size of the knitwork, and puts this into the meta-context of a model of facts and of the inferences which can be applied to them. Second, it picks out the nucleations points of the cloud (or the chunks in the ramen, according to metaphor), and maps them to the entities of an information model. A word of caution: this "picking out of entities" is only simple to the naive; it depends on your ontology (philosophical sense of the theory of things that exist), a question we will tackle in chapters 10 and 11. Close on the heels of the mapping of entities is the mapping of attributes, relationships and so on. Thirdly, the mapping also populates the inference layer, with the expectation that the output of a simple inference in the model will correspond to the result of an intelligent inference in the original knowledge soup.

To be clear (more formal):

- the morphism is a mapping from the knowledge concepts A, B, C, etc. to the information model entities A', B', C';
- it also sets up correspondences from properties A-a, A-b, etc to the information model attributes A'.a', A'.b', etc.
- and the relationships R(A,B) between A and B are mapped to the information model relationships R'(A', B') and so on,
- and where intelligent inference J takes knowledge elements A,B,C... to produce knowledge elements X, Y, Z, then its mapping J' takes entities A', B', C' to produce X', Y', Z', and preserves any property, attribute or relationships.

Conversely, if in the model, we use method J' to infer X' and Y' from A' and B', then we can validate that inference by looking back to the original knitwork, using J to infer X and Y from A and B. That is, we require our morphism to be "knowledge preserving". Or more simply, if we ask our computer a question, then we should get the right answer. The point of going round the houses is to be able to formulate "get the right answer" though a series of simpler, more precise statements - to show this theory has more solid foundations than the bowl of ramen analogy. BTW, there is rather more method to this formulation than meets the untrained eye, but it would be tedious to unpack it here.

Our second morphism seems simple after that: it is a mapping between two knitwork models - entity to entity, attribute to attribute, inference to inference, etc - that also conserves knowledge in so far as it is common to the two models. Or, if you like, a simple, high-level explanation should give results consistent with a more complex and detailed explanation. And this is the morphism we shall be using in level change. The next step is to ground this definition in some examples.

8.4 Grounding 1: Chemical Reactions

In my school chemistry (1960s/70s) [7], there were two sorts of explanation for chemical reactions. At ordinary level, there was "valency", which said that an element would form a particular number of chemical bonds. If Hydrogen forms one bond and Oxygen two, then water had formula H_2O - two oxygen bonds linking to one for each hydrogen atom - imagine little blue hydrogen balls, each with a knob sticking out, and a bigger, yellow oxygen with two holes where the knobs click in.

A second aspect of ordinary level knowledge was the Electro-Negativity sequence, memorised as Kill Nasty Cats, Mangle All Zebras, and so on - Potassium (K) comes before Sodium (Na) which comes before Calcium (Ca). The sequence told you that, for example, sodium would displace an element such as calcium which was lower down the sequence, - mix sodium hydroxide with calcium carbonate to make sodium carbonate and calcium hydroxide.

At advanced level, atoms had a series of electron shells. The number of electrons for the element matched the number of protons in the nucleus, which was given by the element's atomic number. The innermost shell could accommodate two electrons, the next eight and the next 18, and so on. Where the number of electrons did not exactly fill the outermost shell, the atom would swap or share electrons with another atom to complete it. For example, Hydrogen has one electron and only one shell, while Oxygen has eight electrons, two in the inner shell, and six in the outer - both elements having incomplete outer shells. In a water molecule, each hydrogen shared one electron from oxygen's outer shell, giving it two electrons, while oxygen shared the electrons from two hydrogen atoms to fill its outer shell with eight electrons.

Moreover, the columns in the periodic table corresponded to elements having the same number of electrons in the outer shell, and one could infer that if two elements appeared in the same column, then the one with a bigger atomic number came earlier in the electro-negativity sequence - Potassium (element 19) is in the same column as Sodium (11) and is placed earlier in the electro-negativity list. Rubidium (37) is in the same column as Sodium and Potassium, but was not found in the school chemistry lab, so not included in our mnemonic.

When we model this as NQA-knitwork, model both the ordinary and the advanced knowledge using the entity "atom" with attributes *name*, *symbol* and *atomic_number*. In the ordinary model, an *Atom* also has the attribute *valency,* while the advanced model has the inference *Shells(Atom.atomic_number)* which generates the number of electrons in each shell - for example, for Potassium, *Shells(19) -> (2, 8, 8, 1, 0, 0, 0)* (because the outermost shell of an atom can only ever accommodate eight electrons, and therefore the one left over occupies the next shell). For simplicity here, model electro-negativity as an attribute mapping to the electro-negativity series, although in an advanced model it could be inferred from knowledge of the number of shells filled.

The method *BasicTestTubeMetalNitrate(Atom1, Atom2)* is used to infer what happens when a solution of the hydroxide of one metal is mixed the nitrate of another in a test tube - a school chemistry problem. Electro-negativity show what displaces what, and valency gives the proportions.

Use the method to infer the specific result:

$$2KOH + Ca(NO_3)_2 = 2KNO_3 + Ca(OH)_2$$

That is, potassium hydroxide (KOH) mixed with calcium nitrate gives potassium nitrate and calcium hydroxide in the proportions shown. The ordinary level model would use the valencies of potassium and calcium to work this out, while the advanced model could make the inference based on the electron shells of the two elements. The inference that potassium replaces calcium comes from their relative electro-negativities.

That is, there is a mostly straightforward morphism from the advanced level model to the ordinary level one. The advanced level model describes the electron shells, which offer an explanation for the single attribute *valency* in the ordinary level model. Call this morphism a "level change", and specifically one which involves abstraction and relabelling - the electron shell configuration is abstracted and relabelled as "valency".

And say also that the advanced level model, which contains more detail, is at a lower level than the ordinary level model - higher levels corresponding to more abstract descriptions. From the schoolboy viewpoint, the ordinary level model has less to understand - is easier - than the advanced level model. Indeed, even for advanced students it is more convenient to use the ordinary level model for basic reactions than to work "from first principles" with the advanced model. But the advanced level model starts to become more helpful when dealing with more advanced problems. That is, our motive for using a higher level model is that it reduces the mental effort of crunching through the detail that comes with the lower level model, even though it has limitations. And having worked our way through the detail of a low level model of level change, you will probably be relieved that I intend to work through the other examples at a higher level.

8.5 Grounding 2: Transistors to Logic

The propensity of particular elements to gain or lose electrons also underlies the design of semiconductors [8]. Typically, a material such as silicon (valency 4) is doped with a trivalent element such as gallium or a pentavalent element such as phosphorus to change its conductivity, either by donating electrons or accepting electrons. Electronic components such as diodes and transistors are built by creating junctions between electron donors and acceptors. The current flowing through such a junction depends on the bias voltage across the junction - bias in one direction current flows, in the other it is stopped, although to understand properly how this works, we would need to go down a couple of levels into quantum mechanics.

However, rather than going down, we are going up a couple of levels. A transistor has three legs, with the current flowing between two of them being controlled by the voltage on a third. In a transistor amplifier, the control voltage varies continuously, so it continuously varies the output voltage. However, in a logic switch the voltages are quantised to only two values - the on and off voltages. Quantization of the signal means treating it as a number of discrete values. In Transistor-Transistor Logic (TTL) [9] the circuits are designed to work with 0 or 5 volts as the two voltage values.

In logic circuits, On and Off voltages are renamed to True and False, and a circuit which corresponds to a logical operation is termed a gate. For example, an AND gate outputs True only when both inputs are True, otherwise it outputs False, corresponding to the logical operation AND.

In the level change morphism, the entity *Circuit* is mapped to *Gate,* 0v to True and 5v to False (or vice versa).

This also hides some of the knowledge needed to make such systems work. A physical circuit cannot change its output instantly from one value to another, but rather the voltage changes continuously from, say, zero to five volts. In TTL, any voltage between 0.8V and 2V will be "I don't know", but here we are not quantizing to three-valued LPF (cf ch 2), but to classical logic, so such a state must engineered out of the circuit. The practical cheat is to use a clock to co-ordinate circuits, and ignore the change of voltage between the tick and the tock. The clock is also part of the mechanism needed to cascade logic gates - to use the output of one gate as the input to another - and so build up more complex operations.

That is, in the level change morphism, the low level concept *Circuit* is abstracted and relabelled as the higher level concept *Gate,* and voltages to True and False However, the validity of this morphism depends on the careful engineering of the circuits so that the effect of using the electronics is to reproduce logical operations - this is not merely a change of the language used.

In building systems such as computer, this level change - this abstraction - also has the advantage that the same logic functions can be reused from older implementions, such as those using relays and valves. This also has the practical advantage that the engineers who are designing logic-based systems do not have to learn electronics, while the engineers developing individual circuits do not have to understand computer architecture.

8.6 Grounding 3: Logic to Binary

At first glance, the change from the logical values True and False to the binary values 0 and 1 seems to be a trivial relabelling. Moreover, the arrangement of logic gates shown in figure 8.1 is easily shown to be equivalent to a simple device for adding two one bit numbers [8.10].

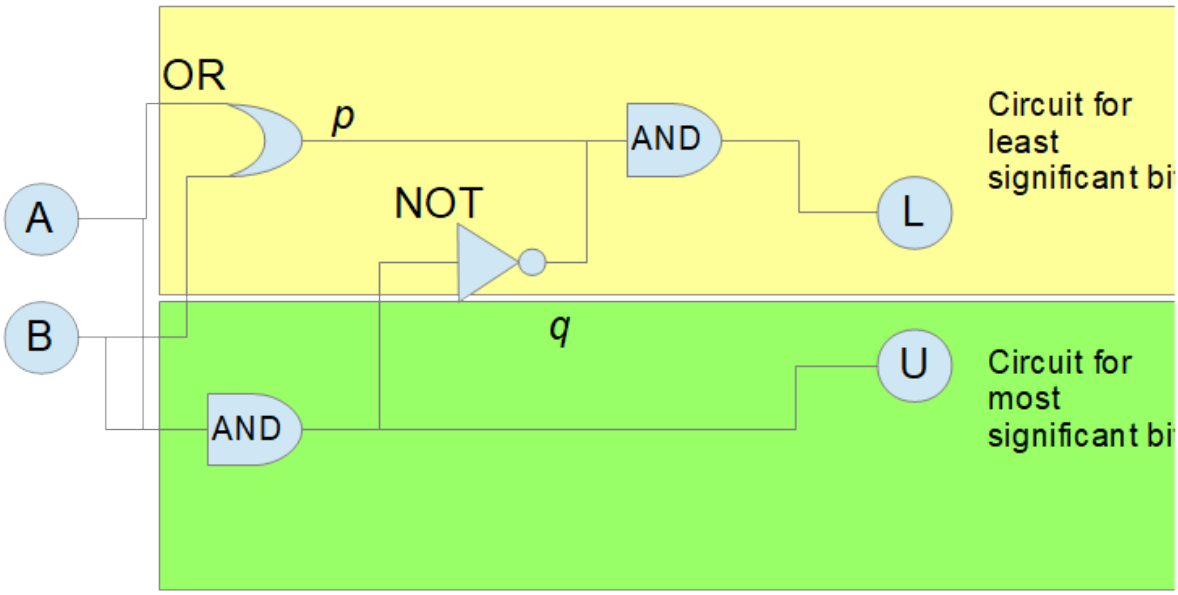

Figure 8.1 Binary Adder from Logic Gates

We can view the Adder circuit as having two component circuits. One for the lower binary digit representing the value 0 or 1, and a second for the higher binary digit, representing 0 or 2. Effectively it uses modulo 2 arithmetic for calculating the low bit, and a simple AND gate for the high order bit. And this points to something more complicated than simply renaming True as 1 and False as zero.

If we set up a collection of 8 lights, then we can count the number of lights that are on we can use these to represent the numbers 0 to 8 (nine different numbers). How ever if we set them out as a row and and treat each light as a digit in an eight-bit binary number, then we can represent the numbers 0 to 255 (256 different numbers). Where did the other 247 numbers come from? The "information" in the two arrangements is different.

The answer lies in how we define the system. In the first, the system (the set of lights) is unstructured - the lights can be counted in any order - while in the second we have added a structure, a type of knowledge - by ordering the bits. In numerical terms, the second bit represents the numbers 0 or 2, the third 0 or 4, and so on, with the binary number being the sum of the values of each of the bits. That is, when going up a level from logic to binary, we have added knowledge to create a system: the knowledge of the ordering of the bits and the inferences that map them to the integers 0 to 255.

There is a woolly thinking step that says that what we put things together, we find the resulting system has "emergent properties", that the numbers 0 to 255 is an emergent property of having eight lights. I would rather say that the set of eight lights is used to design a system with the property that it represents the numbers 0 to 255.

Or, for a less abstract example, consider vehicles travelling along a motor way. In periods of high traffic density one sees "shock waves", which can cause the traffic to slow down and even stop, even though there is no obstruction on the carriage way [11]. At the level of "the individual driver", there is no such thing as a queue of traffic - how can there be when the system boundary stops at the car bonnet? However, when you treat a section of motorway as a whole, there is a system with definite boundaries - the vehicles between the crash barriers - and which contains a large number of interlinked agents. They are not directly linked through their control systems but by the way each driver observes the distance to the car front and follows the rule "don't drive into it". The shock wave is not an emergent property but actually a whole system property - it cannot be inferred from the properties of the individual components acting in isolation.

That is, the level change from logic to binary numbers is not simply one of relabelling. Rather, the system element "binary bit" is now part of a system of binary numbers - a complex world of many components and their relations to each other. It is therefore not surprising that going up a level from logic to binary numbers results in the two levels containing wildly different quantities of information (quantity of information - number of mappings from data to different chunks of knowledge - from lights to numbers).

This implies that going up a level is not necessarily as simple as picking out a subset of the knowledge needed, relabelling it and connecting into a different context. Rather, the context itself links the system into a different, possibly broader, knitwork. Consequently, the term "emergent property" should be treated as a cry of naive surprise, rather than as an explanation. The knowledge

that must be added to the system is not "emergent behaviour" but knowledge of system structure and of whole system properties.

8.7 Not an Example: Open System Interconnect

Open System Interconnect (OSI) [12] is a reference model for the standards needed to interconnect digital systems. Over the years, many, many standards have been developed, and OSI gives a framework in which to pick out a complete and coherent set of them. It splits the interconnection problem into seven layers, starting at the physical layer and working up through layers for data links, networks and so on up to the application layer. Layer or Level?

Start at the physical data layer. The context here is the bit stream, irrespective of what data it conveys or the applications that interpret that data. Its interfaces are upwards to the data link layer and laterally to the system that is being communicated with. The physical layer is only interested in bits and how those bits are physically transmitted to another device in the same layer - if they are transmitted down an optical fibre, then it needs an optical link, if by radio, a radio receiver. The job of the physical layer is to get the bits from one place to another, and what system integrators care about is making sure that my physical layer uses the same standards as your physical layer. If I transmit along wires, then you need to make sure that your wires are connected up in the same way as mine.

Going "up" to the data link layer, the peer-to-peer interface ensures that two nodes start and stop transmissions in the same way, and have compatible flow controls - if my node can only accept a block of 1 MByte in a second, and yours can send at 5 MBytes per second, then flow control might say "transmit one second's worth, then wait another four seconds for me to catch up". The actual mechanics of physically moving the data from your node to my node are dealt with in the layer below - the physical layer. One design goal for data link layer is that it should not affect the contents of message sent. The data link layer interfaces upwards to the network layer, which is responsible for routing the packages of data from your node to mine, rather than to anyone else on the network.

One might think of the OSI model as a weird game of pass-the-parcel, in which two teams of seven players sit either side of a long table. Player one on team A starts with the prize, (the information to be shared) wraps it and passes to player 2, who adds another layer of wrapping before passing it to player 3, and so on to player 7. Player 7 on team A then pass the parcel across to player 7 on team B, who unwraps the outer layer. It then gets passed back up the table towards team B player 1, but with each player taking off one layer of wrapping paper, corresponding to the layer that their opposite number added. In terms of a knowledge model, each player is adding knowledge specific to their layer, but not transforming or remodelling data from any other layer.

The OSI seven layers provides a model that uses layers - like map overlays - to separate out different areas of knowledge. It does not use levels to reformulate knowledge at one layer into a different knitwork for the next layer. Making the connection work takes the knowledge used to wrap the message at each stage going down and matching it to the knowledge needed to unwrap it going up. But it leaves the subject knowledge - the prize - "packaged up", and does not reinterpret it - it does not involve level change.[8.14]

8.8 Is Data to Knowledge a Level Change?

Information is a mapping from something very precise - data - to something relatively vague - a knitwork of facts and inferences. The things we say about data are often quite different to the things we say about knowledge: for example, the number 10 is not rated as trustworthy or old fashioned, whereas a fact - "the Prime Minister lives at number 10" - might be. So is data-to-knowledge a level change in the sense discussed?

Consider the *record* - the computer programming construct which assembles together named data fields plus computing methods (subroutines) for populating the fields and for inferences that can be applied to that record and its fields. At the computing level, records are essentially arbitrary, in the sense the list of fields, the data types used and the inferences are constrained only by the rules of the programming language used. However, a particular record type will be designed to reify an (information model) entity, which is a drop condensed from the cloud of knowledge (or a lump in the ramen).

That is, an information entity is reified as a data record, and the record is designed to "work in the same way" as a particular fact (in context). This is analogous to the way the electronics in a logic gate is designed to work in the same way as a logical operation. Providing we are using entities to represent facts in the right way, and provided the reification is adequate (see chapter 3), then the calculations on the data can be used to infer new knowledge. We can say that the data level and the knowledge level represent the same things, but use different ways of presenting facts and of inferring further facts. We can infer that there are two elephants in the room either by looking into the room and counting, or from adding up the records of the different elephants in our database. Say, therefore, that a *data model* uses a particular subtype of level change, in which the level change is constructed to formalise (at the data level) knowledge of the world modelled.

Note that the level change is between the data level and the knowledge level. The information model appears only as a stepping stone - a mezzanine level - an aid to remembering how data and knowledge are related. In practice, one is likely to program from the information model, or better, its not-quite-a-knitwork form, and in doing so, ensuring that the reification of the information model to data is adequate and accurate. This is because the design of data structures to represent knowledge takes two steps. In the first, the knowledge is condensed from the vague knowledge cloud to be precisely modelled as entities, relations and constraints in the information model. Second, the information model is reified partly by data structures in a programming language, and partly left as constraints and inferences to be implemented by software. That is, in practice, in the computers we have, it is too much work to design physical circuits for every data type used, and it is much easier to fudge the detail using software. This level change involves adding knowledge to the data record, so that we know (at the knowledge level) what the data represents (did I say that knitwork is a recursive concept?)

Indeed, any formal model - a systems model, a process model - involves a translation from the language of daily experience to the formal language of the discipline creating the model. Even logic provides a formal modelling language for reasoning about the world, and any application of classical logic needs to verify that its axioms are applicable in the context it is used - do we assume a particular group of statements to be either True or False? (the excluded middle), or would a multivalued logic such LPF or fuzzy logic be a better fit? Here, one might cry in frustration that some things are just true without the need to map from the modelling language to the real world, but that misses the point of this argument [8.15] - that there is a level change between our model of the world and the world itself. Yes, it is simply true that there is a table in the middle of the room - I

bumped into it on the way in - but because the top is in the shape of Dorset, my AI classified it as a map and so didn't warn me about the table [8.16] - the AI works from a model of the world, rather than physically interacting with it.[8.17]

PS This description gives the counter-intuitive result that a model is at a higher level than the world it models because it is more abstract - has less detail. My initial staircase in chapter 1 was the wrong way up!

8.9 It Works and is Useful. Oh!

What is it about "level change" that makes it a useful tool? It describes a problem at "two levels", such that the lower level supports and justifies inferences at the higher level, while the higher level provides a more focused - more abstract - description that does not get lost in detail. It is a way for humans to simplify their thinking. There is nothing in language itself that means that if an inference is justified at one level is perforce justified at another. Rather, because we can make an inference at one level and then follow the mapping to another level, we can then explore the inference at the second level in the reasonable hope it is justified. And conversely, where the inferences at the two levels do not correspond, level change gives a way into finding why they diverge, and hence identifies errors in our understanding of the two levels. [8.18]

Or - knowledge provides our model of the world. We need to validate our knowledge by seeing how well it predicts the world. Understanding the world at multiple levels is not only cognitively easier, but may give us more easily testable inferences. Understanding electronics helps explain when logic circuits may give the wrong answers, while wrong answers may point to factors not considered in the design of the electronics.

One of the ways we make life more difficult for ourselves is through the flexible way we use language, even when we are not trying to mislead and confuse. In English, we use the term "know" to mean to know a fact, know how to do something and to know a person, while French and Middle English may differentiate between the usages by providing different words - *connaitre* v. *savoir*, or *knowe* v. *can (conne)*. One of the values of level change is the explicit modelling of facts and inferences, which may help show when we have moved from one type of discourse to another - confused similarly named concepts - to show we have flexed when we should have been rigid.

To get to this point, this chapter has inched towards a more formal model of knowledge. The specific sense of level change is identified as a pair of descriptions of knowledge (the two levels) with some facts and inferences in common, together with a form-preserving mapping from one knowledge description to the other (a morphism). This means that what can be inferred in one can also be inferred in the other. The simplest form of level change is the trivial one of relabelling.

A recurring feature of level change is abstraction, where the description at the lower level provides more detail than the higher level. Consequently it takes more effort to use the lower level, where the higher level provides short cuts to the inferences needed. Another feature is system building, where the simple description of the components of a system must be extended by the whole system properties to understand the way the components work together. In these forms of level change, the lower level informs our understanding of the upper level, often because the lower level has been designed to do so.

Another recurring feature is model building - the explicit construction of a language-based model of

the knowledge - here we jump from the embodied world of physical experience to a world constructed from language. Chapter 10 will delve into this distinction further. Say only for the present that we use two different languages, one in which we *talk about* the world, the other in which we *picture* the world, but day to day we use them almost interchangeably and do not notice the level shift from one to the other. I talk about my walking into the table, but do not describe the table, but when I show you a picture of the room, I need to point out that the Dorset-shaped thing in the middle is actually a table, not a map.

And in contrast to level change, layering is a way of dividing up a domain into discrete areas of knowledge. A person can work in a particular layer without having to acquire detailed knowledge of the other layers - to use level change, the person must work explicitly with two formulations of the same knowledge.

BTW, my brilliant Smart Cities proposal didn't get past the internal "agreement to bid" gate, a decision that was mysterious to me, my boss, his boss and even our top technical guy. I took my frustration to another level by taking a secondment to London. The mystery was solved a year later when our building was closed down - my last trip there was for the goodbye photograph. I took early retirement, but, as many before me, I was immediately re-employed as a consultant to continue on my secondment.

Notes and References

8.1. Wikipedia "Smart Cities" https://en.wikipedia.org/wiki/Smart_city accessed 1/11/24 - see particularly the final paragraph of "History" - the bid described was developed c. 2013

8.2. See, for example, "Maps that Made History" Smart L, The National Archives, Kew UK, 2004 ISBN 1 903365 64 3

8.3. Ramen? Just Google it, and you will get pictures and recipes, as well as a Wikipedia entry - or better still, if you are in Japan, go to a restaurant.

8.4. For pictures see, for example, Wikipedia "Telephone Switchboard" https://en.wikipedia.org/wiki/Telephone_switchboard (accessed 1/11/24)

8.5. The geeks might consider an object model (in the OO sense - see, for example, Wikipedia "Object Oriented Programming" https://en.wikipedia.org/wiki/Object-oriented_programming 1/11/24) as explicitly mapping to inferences to object methods, but as we consider generic inferences - logic, arithmetic, Bayesian Belief Networks - then the technicalities of decoupling a multi-use method from a particular object (as an information element) would get in the way.

8.6. See, for example, Wikipedia "Morphism" https://en.wikipedia.org/wiki/Morphism 1/11/24

8.7. References to particular school chemistry textbooks dating from the 1970's would not be helpful here. Just look up anything you don't know.

8.8. From some random Open University programme I watched c. 1967.

8.9. Wikipedia "Transistor Transistor Logic" https://en.wikipedia.org/wiki/Transistor%E2%80%93transistor_logic 2/9/24

8.10 Read the diagram from left to right. In the lower half, inputs from A and B are combined by an AND gate, using the logic that both A and B must be 1 for the most significant but of the binary number to be 1 - for it to represent the number 2. In the upper half, an OR gate combines the inputs to give 1 in the least significant bit if one or both of A or B is true - but this is not exactly what is wanted. Rather if both A and B are 1, then the least significant bit should be 0. The following table illustrates this, where *p* and *q* are the outputs after the OR and the NOT gate respectively.

	A OR B = *p*	A AND B	NOT (A AND B) = q	*p* AND *q* = L
A = 0, B = 0	*0*	0	1	0
A = 0, B = 1	1	0	1	1
A = 1, B = 0	1	0	1	1
A = 1, B = 1	1	1	0	0

8.11. On one occasion, driving out of London on the M4, I was slowed down by such a shock wave. Some ten miles further on I caught up with the cause, a single vehicle travelling at around 40 to 50 miles an hour. Since the typical speeds on a British motorway are between 60 and 80 mph, it meant that everyone travelling in the slow lane pulled out to overtake this vehicle, with the heavy lorries (limited to 60mph) then slowing down everyone doing 70 in the middle lane. However, because of the high traffic density, the people doing 70 rarely had room to pull out into the third lane to overtake, and when they did, they were going slower and in turn slowed down that lane. This was the origin of the shock wave. The mechanisms of its propagation follow from the behaviour of drivers: as they come up to a section of slower traffic, they often slow down slightly too late, and so have to slow down even more not to get to close to the car in front. This amplifies the strength of the shock, and several miles further back, leads to a queue of slow moving traffic.

8.12. Wikipedia "OSI_model" https://en.wikipedia.org/wiki/OSI_model 24/10/24

8.13 As above [8.12], the section "Comparison to other networking suites". The article gives examples of the protocols at each layer, and a comparison with other models of system connection.

8.14 Reminder: definition goes by genus and species. Level change comes under the general heading of knowledge organization, and it differs from other methods of organization by being based on the abstraction and reinterpretation of the same "knowledge in focus". Layering differs from level change by separating out additional knowledge wrapped round the "knowledge in focus" into distinct subdomains.

8.15. This argument comes after Quine.

8.16. I ran into a chap who made county-shaped tables while hitch-hiking in the 1970s.

8.17. There was a video shared on Linked-In of a synthetic view from an autonomous vehicle, in which the thing it thought it saw swapped between a car, a lorry, and a lorry going the other way - the real video showed it was looking at a horse and cart.

8.18. Traditional "proofs of the existence of God" suffer from making strong claims based on logic, but based on the flimsy foundation that statements are either true or false. I had a recursive

argument with a lecturer in philosophical theology as to whether it was true or false that my jeans were blue, to which I replied "It depends whether you use two-valued logic", to which he replied, "Is that true or false?", to which I replied... We terminated the argument with another glass of port.

Bonus Note

The original motive for this discussion was a reply I wrote to Edmund Furse's "Theology of Robots" (*New Blackfriars* Jan 1987) discussing whether computers could have free will. I dismissed the claim that free will would appear "due to level change", but excused myself from justifying my rejection on the grounds that it would take up too much space - it has taken 100 or so pages so far. What I had in mind originally was level-change as relabelling, and I would still reject that idea that causal chains are in any way altered by relabelling.

While an argument could be could be made for free will being a system level property, one way to work out the state of a computer is to construct a second computer and give it all the same inputs. Computers are designed to work consistently, and so two computers should make identical choices - not free will.

For what it's worth, I think that this is the wrong way of posing the problem. Free will is often seen as a *condition* for moral responsibility, since how could we be morally responsible if we had no choice? But I would rather claim that free will is a *consequence* of being told we are morally responsible, since therefore we must choose to understand what is moral and how to act morally. I would further suspect that the combination of individual appetites/drivers and the working through of moral assessments is an intrinsically chaotic process (chaos theory sense), such that we cannot expect individuals to make the same choices - but the key step is giving people free will by telling them that they are morally responsible.